Use of machine-assisted theorem-proving as a means of verifying critical software in the context of RTCA DO 178C

J. Joyce
Draft Version 0.3 (November 30, 2010)

Introduction

RTCA DO 178 (also known as Eurocae ED-12) is used as a basis for the certification of airborne software. This document is also well known beyond the aerospace community as guidance for the development of high assurance software.

A new issue of RTCA DO 178, known as RTCA 178C, is expected to be published by RTCA in 2011. This new issue is expected to include a supplement known as the "formal methods supplement" that provides specific guidance for how verification results obtained by means of formal methods may be used to achieve compliance with RTCA DO 178C.[1]

The formal methods supplement for DO 178C is neutral with respect to the particular kind of formal method used to produce verification results, i.e., it is meant to apply to all kinds of formal methods including abstract interpretation, model checking, theorem-proving and satisfiability solving.

Following the general style of DO 178[2], the formal methods supplement states a number of objectives that must be satisfied by an applicant (i.e., a company or other organization that is seeking certification). For example, the following objective appears in Table FM.A-7 of the supplement:

FM 3	Formal method is correctly defined and sound.	FM 6.2	●	❍	❍	❍	Software Verification Results	11.14	②	②	②	②

This paper is motivated by the possibility that such objectives could be interpreted differently by applicants who use theorem-proving as a means of generating verification evidence for certification purposes. For example, one interpretation of this objective in the context of theorem-proving will require every result to be obtained by means of a finite sequence of inference steps where each step is justified in terms of an inference rule. Alternatively, a less strict interpretation of this objective would also allow approaches to theorem-proving that combine inference rules with complex decision procedures. The possibility of different interpretations is not a shortcoming of the DO 178C formal methods supplement, but rather, a reflection of the fact that the supplement aims to broadly accommodate all approaches to formal methods. The onus is on the community of experts

[1] This paper is based on a draft version of the formal methods supplement.

[2] The term "DO 178" refers in a general way to both RTCA DO 178B (the current version) and RTCA DO 178C (which is expected to soon replace RTCA DO 178B).

Critical Systems Labs
Strategic Insight for Safety

within each area of formal methods, such as the theorem-proving community, to clarify how such objectives should be addressed.

The immediate aim of this paper to stimulate a discussion of how specific objectives of the DO 178C formal methods supplement should be refined and elaborated in the context of theorem-proving approaches. An ideal outcome of this discussion would be a protocol that is recognized by a critical mass of qualified experts as an acceptable way to use theorem-proving to generate verification results for high assurance systems.

The remaining of this paper is organized in terms of a series of questions motivated directly by particular aspects of the RTCA DO 178C formal methods supplements.

Question #1: What should be required to accurately identify a particular approach to theorem-proving?

The above question is motivated by the several elements of the DO 178C formal methods supplement including:

> FM 1.6.1: In the context of this document, to be *formal*, a model must have an unambiguous, mathematically defined syntax and semantics
>
> FM 4.3: The Software Verification Plan will detail the methods used to gather the verification evidence including that obtained by *formal analysis*.
>
> FM 6.2: All notations used for *formal analysis* should be verified to have precise, unambiguous, mathematically defined syntax and semantics i.e., they are *formal notations*.
>
> FM 11.6: Notations used to define *formal models* must have mathematically defined syntax and semantics.

It is reasonable to expect that the theorem-proving approach used to generate verification results is identified with sufficient detail to ensure that a qualified person independent of the verification effort, such as a certification authority, could determine exactly what logical formalisms and supporting tools have been used. The logical formalisms should be identified independently of the supporting tools, i.e., it is not sufficient to rely on the functionality of a software tool as a description of the logical formalism.

Adequate identification of the theorem-proving approach will help to avoid the situation where there is uncertainty about the formalisms used to produce the verification results or uncertainty about which tools were used to produce these results. Achieving this objective should also help avoid situation in which a particular approach is inaccurately described as a theorem-proving approach, e.g., the verification tool is not based on any formal system of logical inference.

At least part of the answer to Question #1 is likely to refer to the need for a written description of the formal syntax and formal semantics of the formalism underlying the theorem-proving approach, including the axioms and inference rules. If the formalism is a "typed" logic, it is also reasonable to expect a written description of a positive effective test for determining that an expression is "well-typed".

A complete answer to Question #1 should also identify all of the software tools involved in the generation of verification results using theorem-proving, including tools whose purpose is to verify the verification results, e.g., proof-checkers.

If a theorem-proving approach relies on decision procedures beyond the application of inference rules to generate verification results, these decision procedures should be identified and described.

Some theorem-proving approaches rely on some previously developed infrastructure that is not part of the logical formalism, but may still be considered "built-in" into the formalism. For example, in the HOL system there are previously developed "theories" for a wide variety of abstractions that can be imported into a particular use of HOL. An example is a theory that provides definitions of types and constants for elements of various abstract data types such as sets and lists. Importing this theory would also provide access to various previously proved theorems. It seems reasonable to expect that such extensions of the formalism need to be identified.

Question #2: What should be required to demonstrate that a particular theorem-proving approach is sound?

This question is motivated by the several elements of the DO 178C formal methods supplement including:

> FM 1.6.2: Any tool that supports the formal analysis should be assessed under the tool qualification guidance required by DO-178 and qualified where necessary.
>
> FM 6.2: The soundness of each *formal analysis* method should be justified. A sound method never asserts that a *property* is true when it may not be true.

It is reasonable to expect that verification results obtained by means of theorem-proving can be trusted with a very high level of confidence, and that this trust is based on objective analysis. The usage history of a particular theorem-proving approach (e.g., "… it has never produce a false result so far …") or quality-based factors (e.g., "… the theorem-proving tools have been implemented by highly experienced software developers …") may be useful as a supplementary evidence for the soundness of a theorem-proving approach, but should not be relied upon as primary evidence for satisfaction of this objective. The soundness of both the formalism(s) and the implementation of the software tool(s) needs to be addressed. Moreover, the soundness of any assumptions that are used to produce verification results

Critical Systems Labs
Strategic Insight for Safety

must also be demonstrated. For example, these may be assumptions about the physical environment, e.g., an airplane cannot be simultaneously "airborne" and "landed".

One possible approach to demonstrating the soundness of software tools used to generate verification results by means of theorem-proving is to rely on a separate "proof-checker" that simply verifies that a proof generated by another tool is indeed a proof based on the criteria of the underlying formalism, e.g., every line in the proof is an axiom or follows from a previous line of the proof by means of an inference rule. In this case, it would seem adequate to only demonstrate the soundness of the proof-checker. This is an attractive solution because the proof-checker is likely to be a relatively simple function compared to the proof generation tool. However, this approach requires the proof generation tool to generate a proof as an output which, in turn, can be used as input to the proof-checker. This could be problematic in the case of a theorem-proving approach that relies heavily on decision procedures that may not create an explicit proof of its results.

Question #3: Is the notion of "conservative representation" sufficient for a theorem-proving approach to ensure that the formalization of a verification problem does not admit false results?

An important concept of the DO 178C formal methods supplement is the notion of "conservative representation":

> FM 6.3: Requirement formalization correctness: If a requirement has been translated to a formal notation as the basis for using a formal analysis, then review or analysis should be used to demonstrate that the formal statement is a conservative representation of the informal requirement.
>
> FM Annex B (Glossary): Conservative representation - A simplified or abstract representation in a formal notation of life-cycle data (such as requirements or code) such that statements that are true for this representation will always be true for the life-cycle data itself. In many cases, analysis of a conservative representation will yield pessimistic results but never unsound ones. The representation chosen simplifies or makes possible the analysis

The intent of the above elements of the formal methods supplement is to ensure that the process of formalization does not admit the possibility of false verification results.

In general, confirmation that a formal model is a conservative representation of some "life-cycle data" is not something that can be demonstrated formally (unless the "life-cycle data" is itself a formal model).

A typical use of theorem-proving to verification is a matter of using formal proof to demonstrate that the requirements of one level of description, such as a formalization of the requirements, is satisfied by another "lower level" description such as a formal model of the design. A common approach is to generate a theorem of the form, $\vdash D \rightarrow R$, where D and R denote the formal model of the design and the formalized requirement respectively.

Clearly, it is desirable to require *D* to be a conservative representation of the actual implementation. However, what is the desired relationship between R and the "actual" requirements which, for example, might be expressed by a sequence of sentences written in English? The constant "true" is, by definition, a conservative representation of everything. Obviously it is pointless to use theorem-proving to produce ├ D→ R as a verification result if R is logically equivalent to "true". However, there is a possibility that it may not be apparent that R is logically equivalent to "true". More generally, there is a possibility that R may "mean less"' than what is meant by the actual requirements. This possibility is not mitigated by assurance that R is a conservative representation of the actual requirements.

Another facet of this question is the manner in which verification results are expressed in a theorem-proving approach. The generation of a theorem of the form ├ $D \rightarrow R$ is just one possible way of expressing a verification result. Another possibility is equivalence, i.e., ├ $D \leftrightarrow R$. *Yet another possibility is* ├ $(D \neq false)$ & $(D \rightarrow R)$, which is meant to guard against the possibility of the so-called "false implies everything" problem in the case of a logically inconsistent formal model. Would it be reasonable to recommend certain patterns or templates for the theorems generated for the purposes of certification?

Question #4: Should there be limitations on the number and complexity of requirements addressed by a single verification result obtained by means of theorem-proving?

Traceability between requirements and verification results is a fundamental value of the best practices for development of high assurance software across many industries. The granularity of this traceability is often very fine, especially in the case of software component associated with high risk. A typical deliverable would be a matrix that maps individual "shall statements" to specific test cases and the corresponding results of executing these test cases. This expectation of traceability also applies to requirements that must be verified by means other than test, e.g., inspection or analysis. The absence of this mapping from requirements to verification results would likely be a serious obstacle to certification.

Is it reasonable to expect the same degree of granularity in the relationship a mapping from requirements to verifications results in the case where these verification results are obtained by means of theorem-proving?

One problem in satisfying this well-established expectation of traceability occurs when the formalization of a set of requirements is not merely the translation of individual "shall statements" one by one into a formal expression. Instead, the formalization might be a formal model that is intended to represent a set of requirements without any direct mapping of specific elements of the formal model to specific requirements. This more "holistic" approach might in fact yield a better result than an attempt to translate individual "shall statements" into formal expressions. However, such an approach would not easily accommodate traceability from individual requirements to verification results, or at least, it would be much more difficult to explicitly document such traceability at the fine level of granularity normally expected for the purposes of certification.

Critical Systems Labs
Strategic Insight for Safety

In general, traceability serves as a check that every element of the input to a particular step in development has been fully developed in the output of this step. Traceability also serves as a check that every element of the output from this step is necessary with respect to the input.

Several objectives of DO 178 express very specific expectations for traceability, e.g., traceability between system requirements and high-level software requirements, traceability between high-level software requirements and low-level software requirements, traceability between low-level software requirements and source code, traceability between source code and executable object code.

Some (but not necessarily the author of this paper) might argue that traceability is merely an crude check on the output of a development step which partially compensates for the shortcomings of test-based verification in a conventional approach to software development – and as such, the use of formal analysis makes traceability redundant. Some might even say that "traceability comes for free" when formal methods are used. Nevertheless, the formal methods supplement for DO 178C recognizes the importance of traceability with the following guidance:

> FM 6.3: Traceability: Traceability ensures that all input requirements have been developed into lower level requirements or source code. Traceability from the inputs of a process to its outputs can be demonstrated by verifying with a *formal analysis* that the outputs of the process fully satisfy its inputs. Traceability from the outputs of a process to its inputs can be demonstrated by verifying with a *formal analysis* that each output data item is necessary to satisfy some input data item.

In the case of a theorem-proving approach to generating verification results for the purposes of certification, what is a reasonable expectation for the granularity of traceability from requirements to verification results?

It is easy to imagine a scenario in which a formal model, *R*, of the required behaviour has been created to simultaneously represent several dozen "shall statements" and theorem-proving used to generate prove a single theorem of the form ├ $D \rightarrow R$. The input to the software tool might be the desired theorem, $D \rightarrow R$, and the output might simply be an indication that the desired theorem has been successfully proven, e.g., "Theorem proved" or perhaps a statement of the generated theorem, ├ $D \rightarrow R$. For such a scenario, the only traceability that might be documented is a record a mapping from each of the input requirements to the same verification result, e.g., "Theorem proved". This would contrast sharply with the concept of traceability in a conventional test-based approach in which most input requirements are mapped to specific verification results that exist specifically for the purpose of verifying that particular requirement.

In the case where theorem-proving is used to generate verification results for the purpose of certification, it may be desirable to impose a limit on the number and complexity of requirements that are addressed by a single verification result. While this may not improve

the thoroughness of the verification results, it may be necessary to impose such limits to make these results acceptable to certification authorities.

Beyond the goal of ensuring that verification results are presented in a form acceptable to certification authorities, some limitations of the number and complexity of requirements addressed by a single verification result might also be desirable as protection against the possibility that the formalization of a set of requirements is incomplete. The possibility that the formalization of a set of requirements is incomplete is higher when there is not a one-to-one relationship between individual requirements and elements of the formalization.

Objective 5: How should verification results obtained by means of theorem-proving be documented?

It is reasonable to expect that verification results obtained by means of theorem-proving are documented with sufficient detail that they can be fully understood and interpreted by an qualified expert without access to the person(s) who originally performed the work that produced these results, without access to other data created when the work was performed and without access to the software tools used to generate the results. In short, the verification results must be documented in a manner that encapsulates everything that a qualified expert would need to confirm the validity of the results.

The formal methods supplement for DO 178C specifically addresses the need for the documentation of verification results:

> FM 6.3.8: Formal analysis cases: The objective is to provide evidence that the formal analysis achieves the coverage required for the applicable software level. Any assumptions that were included in the formal analysis should be justified and any false assumptions (which would invalidate the analysis) should be identified and removed. The verification of analysis cases is presented in paragraph FM.6.5.
>
> FM 6.3.8: Formal analysis procedures: The objective is to verify that the formal analysis cases were accurately developed into formal analysis procedures and expected results.
>
> FM 6.3.8: Formal analysis results: The objective is to ensure that the formal analysis results are correct and that discrepancies between actual and expected results are explained.
>
> FM Annex B: Formal analysis case - a combination of the property to be analysed or proved, all assumptions for the formal analysis, and expected results, such as to verify correct functionality and to establish conditions that reveal potential errors; analogous to a test case.

Clearly, something more detailed than the message "Theorem Proved" needs to be recorded to serve as the result of a verification effort. At the very least, the result needs to

Critical Systems Labs
Strategic Insight for Safety

identify assumptions that the verification result may rest upon. Understanding a verification result obtained by means of theorem-proving is also likely to depend on a potentially large set of definitions created during the formalization process. How should these definitions and any other supporting information be bundled up with the derived theorems?

Another aspect of this question is whether the verification result obtained by means of theorem-proving should be a formal proof, or whether there verification result is merely the theorem whose existence is established by the formal proof?

Objective 6: How can theorem-proving demonstrate the absence of unintended functionality?

For high assurance systems, an important part of the verification effort is to demonstrate freedom from intended functionality. In a conventional test-based approach, this is often achieved by a combination of traceability and structural coverage analysis. The formal methods supplement of DO 178C articulates this particular aspect of verification in terms of an obligation to show "each output data item is necessary to satisfy some input data item" in reference to individual steps in the software development process.

One way to demonstrate the absence of unintended functionality is to prove that a design is logically equivalent to a set of requirements, i.e., $\vdash D \leftrightarrow R$ *instead of just* $\vdash D \rightarrow R$. However, proving equivalence might not a practical strategy for non-trivial systems.

Are there are other standard ways in which freedom from unintended functionality could be demonstrated using theorem-proving? For example, would it be reasonable to demonstrate that an output data item is necessary by proving that the negation that an assertion which contradicts the meaning of the output data item also contradicts the input requirements? If so, could this approach be "standardized" in a manner that could be widely used across a variety of theorem-proving approaches that are used for the purpose of generating verification evidence for certification?

Summary

The formal methods supplement of RTCA DO 178C is a significant opportunity for the theorem-proving community to integrate theorem-proving approaches into the mainstream of companies and other organizations who develop and certify airborne software. However, uncertainty about the interpretation of certain objectives contained in the formal methods supplement, or a lack of consistency in how these objectives are interpreted, could undermine the acceptance of theorem-proving approaches as a means of verifying airborne software. This short paper has identified a number of questions for consideration by the theorem-proving community about the interpretation of the formal methods supplement. It hoped that these questions will stimulate a discussion that eventually results in a protocol for how theorem-proving can be used as a means of generating verification evidence that is suitable for purpose of certifying high assurance systems.